สร้างภาพเคลื่อนไหว ง่ายดั่งพลิกฝ่ามือ

ศิวะ วงศ์ตุรัน

หลายท่านอาจจะเคยใช้ภาษาคอมพิวเตอร์ เช่น BASIC, PASCAL หรือ C ในการเขียนโปรแกรมคอมพิวเตอร์กับงานพวกสร้างภาพในกราฟิก สิ่งที่ภาษาคอมพิวเตอร์เหล่านี้ให้มาก็คือ คำสั่งที่ช่วยเขียนวงกลม-วงรี-เส้นตรง-ระบายสี ฯลฯ คำสั่งเหล่านี้ช่วยให้งานสะดวกขึ้นมากทีเดียว แต่น่าเสียดายคือไม่มีคำสั่งทำภาพเคลื่อนไหวโดยเฉพาะให้เราได้ใช้งานบ้างเลย ถึงจะสร้างภาพเคลื่อนไหวสักภาพจะต้องเขียนโปรแกรมเอง แต่ต่อไปนี้การสร้างภาพเคลื่อนไหวจะง่ายขึ้นมากถ้าท่านได้รู้จักกับ FM.COM

อะไรคือ FM.COM

โปรแกรม FM.COM (FAST MOVE IMAGE) คือ โปรแกรม TSR ที่พร้อมจะถูกเรียกใช้งานได้ตลอดเวลา ตัวโปรแกรมเขียนด้วย MACRO ASSEMBLER VERSION 5.0 มีหน้าที่ทำให้ภาพบนจอภายในกรอบและตำแหน่งที่กำหนดไว้เคลื่อนไหวในแนวเส้นตรง โดยการสั่งงานจากโปรแกรมของผู้ใช้ผ่านทางอินเตอร์รัพต์เบอร์ 6 โปรแกรมที่เรียก FM.COM นั้นจะเป็นภาษาใดก็ได้ที่สามารถสั่ง INT และเขียนข้อมูลลงในหน่วยความจำในตำแหน่งที่กำหนดได้ อันที่จริงส่วนใหญ่การเรียกใช้บริการอินเตอร์รัพต์นั้นมักจะมีการผ่านข้อมูลไปทางรีจิสเตอร์ การผ่านข้อมูลโดยวิธีนี้มีข้อดีคือ ทำงานได้รวดเร็ว แต่ผ่านข้อมูลได้น้อยและมักจะต้องเขียนข้อมูลลงในรีจิสเตอร์ใหม่ทุกครั้งที่มีการเรียกใช้ ส่วนวิธีผ่านข้อมูลทางการใช้หน่วยความจำจะเหมาะถ้ามีข้อมูลค่อนข้างมาก อีกทั้งไม่จำเป็นต้องเขียนข้อมูลทั้งหมดใหม่ทุกครั้งที่เรียกใช้บริการอีกด้วย ผู้เขียนจึงเลือกใช้วิธีนี้ในการผ่านข้อมูลไปยัง FM.COM ส่วนลักษณะการเคลื่อนที่ของภาพนั้นแสดงดังรูปที่ 1

สำหรับข้อมูลที่จำเป็นจะต้องเขียนลงหน่วยความจำประกอบด้วย

- จุดเริ่มต้น (ที่มุมบนซ้าย) ของกรอบภาพเคลื่อนไหว (X1, Y1)
- จุดสิ้นสุด (ที่มุมบนซ้าย) ของกรอบภาพเคลื่อนไหว (X2, Y2)
- ความเร็วของภาพเคลื่อนไหวที่ต้องการ (SPEED)
- ความกว้างของภาพ (ในแนว X)
- ความสูงของภาพ (ในแนว Y)

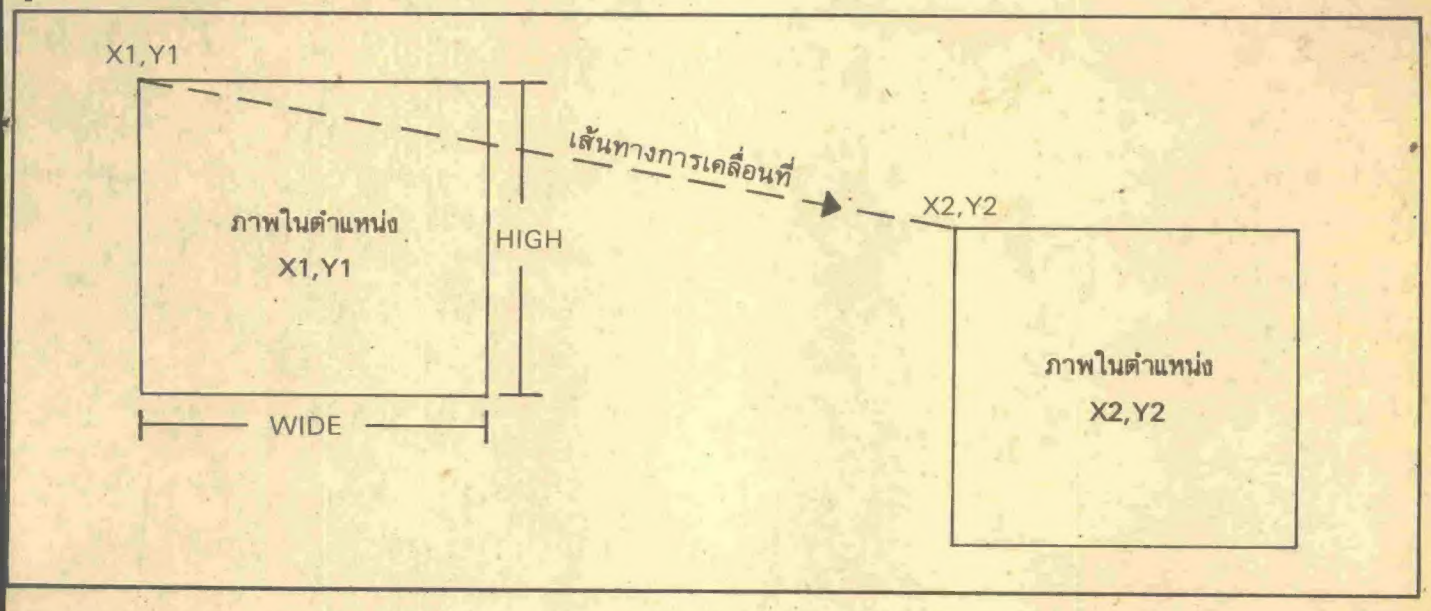
ผู้เขียนมองภาพเคลื่อนไหวเหมือนภาพที่บรรจุในกรอบสี่เหลี่ยมที่มีความกว้างคิดเป็นจำนวนไบต์และความสูงคิดเป็นจำนวนจุดของภาพ โดยภาพในกรอบจะเคลื่อนไหวในแนวเส้นตรงจากจุดเริ่มต้น (X1, Y1) ไปยังจุดสิ้นสุด (X2, Y2) ระยะเวลาที่ใช้ไปกำหนดด้วยความเร็วของภาพ (SPEED) ลักษณะการเคลื่อนไหวนั้น ภาพในกรอบมีลักษณะเป็นภาพนิ่งเท่านั้น เมื่อผ่านฉากหลังจุดภาพแต่ละจุดไม่ทำให้ฉากหลังเสียหาย

ข้อกำหนดเกี่ยวกับข้อมูลที่ส่งผ่านไปยังหน่วยความจำมีดังนี้

X1 + (WIDE * 8) = < 719
Y1 + HIGH = < 347
X2 + (WIDE * 8) = < 719
Y2 + HIGH = < 347
SPEED มีค่าได้ตั้งแต่ 1 ถึง 0FFFFH
HIGH * WIDE มีค่าอยู่ระหว่าง 1 ถึง 2000H

ภาพในกรอบทางด้านความกว้าง (WIDE) คิดจำนวนเป็นไบต์ เช่น จากจุด X=0 ถึง X=50 กำหนดความกว้าง = 7 (8*7 = 56 จุด) ความสูงของภาพคิดเป็นจำนวนจุดภาพในแนวแกน Y ความเร็วของภาพต่ำสุดคือ 0FFFFH สูงสุดคือ 1 สิ่งสำคัญในการกำหนดขนาดของกรอบภาพคือให้กำหนดใช้น้อยที่สุดเท่าที่จำเป็นเท่านั้น เนื่องจากภาพขนาดใหญ่ความเร็วลดลงมาก ส่วนค่า SPEED ถ้าใช้ตัวเลขมาก ๆ การเคลื่อนที่จะนิ่มนวลกว่าแต่เคลื่อนได้ช้า ทั้งหมดนี้อยู่ในการตัดสินใจของผู้เรียกใช้ เท่าที่ทดลองกับเครื่อง IBM PC ความถี่ 4.77 MHz ธรรมดา ๆ พบว่าความเร็วที่ได้ยังอยู่ในขั้นที่น่าพอใจ จึงเชื่อว่าในเครื่องรุ่นอื่นที่มีความเร็วสูงกว่านี้ ภาพเคลื่อนไหวที่ได้ย่อมรวดเร็วทันใจผู้เรียกใช้อย่างแน่นอน

รูปที่ 1 แสดงลักษณะการเคลื่อนที่ของภาพ



ขั้นตอนการติดตั้งและใช้งาน

เริ่มต้นจากเมื่อเข้าสู่ดอสให้พิมพ์ FM.COM ตามด้วย ENTER ไม่เกิน 10 วินาที ภาพโฆษณาโปรแกรมก็ปรากฏขึ้น หลังจากนั้นเป็นหน้าที่ของการเรียกโปรแกรมที่ผู้ใช้งานเขียนไว้ด้วยภาษาคอมพิวเตอร์ทั้งหลายแหล่เข้ามาสั่งงาน FM.COM ตามใจชอบ FM.COM จะอยู่ในหน่วยความจำจนกว่าจะปิดเครื่อง มีสิ่งที่ต้องทำความเข้าใจในการเรียกใช้งานอยู่ก็คือ

- FM.COM ทำงานกับโมโนโครมกราฟิก เช่น เฮอร์คิวลิส เท่านั้น

- ก่อนสั่ง INT 6 จะต้องอยู่ในโหมดกราฟิก เพจ 0 (PAGE 0)

วิธีการส่งผ่านข้อมูลมีขั้นตอนดังต่อไปนี้

- หาค่า SEGMENT : OFFSET ของ INT 6 เพื่อหาตำแหน่งหน่วยความจำที่จะส่งผ่านข้อมูลเสียก่อน

- ค่าของ OFFSET+4 ที่หาได้คือค่าของตำแหน่งเริ่มต้นที่จะส่งผ่านข้อมูล

รูปแบบการเรียงลำดับของข้อมูลเป็นดังนี้

OFFSET + 4	= X1	ขนาดข้อมูลคือ	2 ไบต์
+ 6	= Y1	ขนาดข้อมูลคือ	2 ไบต์
+ 8	= X2	ขนาดข้อมูลคือ	2 ไบต์
+ 10	= Y2	ขนาดข้อมูลคือ	2 ไบต์
+ 12	= SPEED	ขนาดข้อมูลคือ	2 ไบต์
+ 14	= WIDE	ขนาดข้อมูลคือ	1 ไบต์
+ 15	= HIGH	ขนาดข้อมูลคือ	1 ไบต์

ที่ควรระวังคือ เรื่องการคำนวณตำแหน่งผิดพลาด เหตุการณ์เช่นนี้ถ้าเกิดขึ้น อาจจะทำให้โปรแกรมของท่านทำอะไรแปลก ๆ อย่างที่คาดไม่ถึงออกมาได้ แต่คิดว่าตัวอย่างของโปรแกรมเรียกใช้ FM.COM ซึ่งผู้เขียนได้ทำไว้คือโปรแกรมแรกเป็นการเรียกใช้จากภาษา BASIC และโปรแกรมที่สองเป็นการเรียกใช้ภาษา C นั้นคงจะช่วยให้ท่านเข้าใจได้เป็นอย่างดี

ใช้โปรแกรม BASIC

ในส่วนนี้ผู้เขียนใช้ BASIC รุ่นเก่า ๆ (โปรแกรมแสดงดังรูปที่ 2) คือ HBASIC มาเป็นตัวอย่าง ถ้าท่านใช้ TURBO BASIC หรือ QUICK BASIC โปรแกรมคงง่ายขึ้นอีกเยอะ ในการติดต่อกับ FM.COM จะสังเกตจากโปรแกรม HBASIC ได้ว่าไม่มีคำสั่ง SCREEN เพราะ HBASIC อยู่ในกราฟิกตลอดเวลาจึงไม่จำเป็นต้องใช้ สิ่งที่แย่สำหรับ HBASIC หรือ BASIC รุ่นเก่า ๆ ก็คือ คำสั่ง POKE ส่งข้อมูลได้ครั้งละ 1 ไบต์ แต่ FM.COM ต้องการข้อมูลครั้งละ 2 ไบต์ ทำให้ยุ่งยากขึ้นอีกเยอะทีเดียว ส่วนของ

```
100 CLEAR 26676!
110 X1H% = 0:X1L% = 2
120 Y1H% = 0:Y1L% = 2
130 X2H% = &H2:X2L% = &H6F
140 Y2H% = &H1 :Y2L% = &H2
150 SPEEDH% = &H88:SPEEDL% = &H88
160 W% = 12
170 H% = 90
180 CLS
190 LINE(2,2)-(90,89),,B:CIRCLE (46,45),40,1,,,2/3:PAINT (46,45)
200 FOR I% = 1 TO 5
210 READ CODE%
220 POKE (26675! + I%),CODE%
230 NEXT I%
240 PASTMOV = 26676!:DEF SEG = 0
250 MSEG = 256 * PEEK(6*4 + 3) + PEEK(6 * 4 + 2)
260 MSE  = 256 * PEEK(6*4 + 1) + PEEK(6 * 4) + 4
270 DEF SEG = MSEG
280 POKE MSE  ,X1L%:POKE (MSE + 1),X1H%
290 POKE (MSE  + 2),Y1L%:POKE (MSE + 3),Y1H%
300 POKE (MSE  + 4),X2L%:POKE (MSE + 5),X2H%
310 POKE (MSE  + 6),Y2L%::POKE (MSE + 7),Y2H%
320 POKE (MSE  + 8),SPEEDL%:POKE(MSE + 9),SPEEDH%
330 POKE (MSE + 10),W% :POKE (MSE + 11),H%
340 DEF SEG
350 CALL PASTMOV
360 END
370 DATA &h55,&hcd,&h06,&h5d,&hcb
```

รูปที่ 2 โปรแกรม BASIC ที่เขียนเพื่อเรียกใช้ FM.COM

DATA นั้นตัวเลขฐานสิบหกที่มีอยู่เป็นคำสั่งภาษาเครื่องแปลได้ใจความดังนี้

```
PUSH BP
INT 6
POP BP
IRET
```

ในส่วนการหาค่า SEGMENT:OFFSET นั้นดูได้จากบรรทัด 250-260 ค่า MSEG ก็คือค่า SEGMENT และ MSE คือค่า OFFSET อันที่จริงแล้วตารางอินเตอร์รัพต์มีค่าคงที่แน่นอนตายตัวไม่จำเป็นต้องคำนวณก็ได้ ขอให้ท่านลองสังเกตจากโปรแกรม C

ใช้โปรแกรม C

จะเห็นว่าในโปรแกรม C เราเรียกเอาค่า SEGMENT ของ INT 6 จากค่า 0:0×1A โดยตรงเลย เช่นเดียวกับการเรียกค่า OFFSET จากค่า 0:0×18 หลังจากนั้นก็ผ่านข้อมูลต่าง ๆ ลงไปโดยเริ่มจากตำแหน่งแรกคือ OFFSET_VALUE +4 เรื่อยไป ท้ายสุดจึงใช้ฟังก์ชัน GENINTERRUPT เรียก INT 6 ทำงาน โปรแกรมแสดงดังรูปที่ 3

รูปที่ 3 โปรแกรม C ที่เขียนเพื่อเรียกใช้ FM.COM

```
#include <graphics.h>
#include <dos.h>
int getch(void);

/* PROGRAM FOR  CALL  FAST_MOV_IMAGE */

main()
  {
  int  seg_value, offset_value;
  int gd = DETECT, gm;

  seg_value = peek(0,0x1a);
  offset_value = peek(0,0x18);
  offset_value = offset_value + 4;

  poke(seg_value,offset_value,0);          /* X1 */
  poke(seg_value,offset_value + 2,0);      /* Y1 */
  poke(seg_value,offset_value + 4,500);    /* X2 */
  poke(seg_value,offset_value + 6,200);    /* Y2 */
  poke(seg_value,offset_value + 8,0x8888); /* SPEED TO MOV IMAGE */
  pokeb(seg_value,offset_value + 10,12);   /* WIDE OF IMAGE (BYTE) */
  pokeb(seg_value,offset_value + 11,90);   /* HIGH OF IMAGE (DOT)  */
  initgraph(&gd, &gm, "");
  outtext("          *** PRESS ANY KEY TO MOV IMAGE  AND EXIT *** ");

  rectangle(0,0,89,89);
  circle (44, 40,39);
  floodfill(44,40,1);
  getch();
  geninterrupt(6);
  getch();
  closegraph();
  return(0);
  }
```

ในการเรียกใช้ FM.COM จากภาษาอื่นก็คงมีขั้นตอนเช่นเดียวกัน โปรแกรมตัวอย่างทั้งสองนี้เขียนขึ้นอย่างง่าย ๆ เพื่อความเข้าใจ สำหรับการเรียกใช้ FM.COM อย่างจริงจังนั้นสามารถเขียนให้ยืดหยุ่นกว่านี้ได้อีกมาก ทั้งนี้ขึ้นอยู่กับผู้พัฒนาโปรแกรม ข้อมูลที่ส่งผ่านไปนั้นไม่ว่าจะมาจากการใช้ภาษาใดก็ตาม จะยังอยู่คงเดิมตลอดเวลา จนกว่าเมื่อไหร่ที่เขียนข้อมูลใหม่ลงไปทับนั่นแหละจึงจะเป็นการเปลี่ยนแปลงข้อมูล ดังนั้นไม่จำเป็นต้องเขียนข้อมูลทั้งหมดลงไปใหม่ทุกครั้ง เช่น เราต้องการค่า SPEED=0AAAAH ตลอดเวลาที่เรียกใช้ ก็เพียงเขียนค่า 0AAAAH ครั้งเดียวเป็นอันเสร็จสิ้น

เบื้องหลังของ FM.COM

สาเหตุที่กล่าวถึงเบื้องหลัง FM.COM คงมีเพียงอย่างเดียวคือ เสนอแนวความคิดในการพัฒนาโปรแกรมแก่ท่านผู้สนใจได้นำไปประยุกต์ใช้ หรืออาจจะมีแนวความคิดใหม่ ๆ เกิดขึ้นได้มาก และถ้าเป็นจริงเช่นนั้นก็นับว่าเป็นการคุ้มค่าทีเดียว ส่วนตัวโปรแกรมนั้นแสดงดังรูปที่ 4

รูปที่ 4 ซอร์สโค้ดของโปรแกรม FM.COM

```
page 58,132
HERC SEGMENT  PARA
ASSUME CS:HERC, DS:HERC, ES:HERC
                ORG 100H
    START:
                CLD
                MOV AX,3506H   ;get interrupt vector
                INT 21H
                MOV AX,ES:[BX + 2]
                CMP AX,01234H
                JNZ SHORT INSTALL
                MOV DX,OFFSET REPEAT_RUN
                MOV AH,09H
                INT 21H
                MOV AH,4CH
                INT 21H

    INSTALL:    MOV DX,03B8H
                MOV AL,00100000B   ;off screen
                OUT DX,AL
                MOV DX,OFFSET TITLE_
                MOV AH,09H
                INT 21H
                MOV AL,00101000B   ;on screen
                MOV DX,03B8H
                OUT DX,AL
                MOV AX,DS
                MOV ES,AX
                MOV AX,0100H
                MOV DI, OFFSET TABLEX
    XDAT2:      ROR AH,1
                MOV DL,AH

                CMP AH,80H
                JNZ SHORT CL7
                MOV AH,8
                JMP SHORT CLX

     CL7:       CMP AH,40H
                JNZ SHORT CL6
                MOV AH,7
                JMP SHORT CLX

     CL6:       CMP AH,20H
                JNZ SHORT CL5
                MOV AH,6
                JMP SHORT CLX

     CL5:       CMP AH,10H
                JNZ SHORT CL4
                MOV AH,5
                JMP SHORT CLX

     CL4:       CMP AH,8
                JNZ SHORT CL3
                MOV AH,4
                JMP SHORT CLX

     CL3:       CMP AH,4
                JNZ SHORT CLX
                MOV AH,3
     CLX:       XCHG AH,AL
                STOSW
                XCHG AH,AL
                MOV AH,DL
                CMP DI,1442 + OFFSET TABLEX
```

```
                JGE SHORT XDAT1
                CMP AH,1
                JNE SHORT XDAT2
                INC AL
                JMP SHORT XDAT2

XDAT1:          MOV DI,OFFSET TABLEY
                XOR CX,CX
                XOR DH,DH
                MOV BX,045AH
YTAB:           MOV AX,CX
                DIV BH
                MOV DL,AH
                MUL BL
                MOV SI,AX
                MOV AX,2000H
                MUL DX
                ADD AX,SI
                STOSW
                INC CX
                CMP CX,348
                JL  SHORT YTAB

                MOV AX,2506H   ;SET interrupt vector
                MOV DX,OFFSET START_MOV
                INT 21H
                MOV AX,01234H
                MOV BX,DX
                MOV [BX + 2],AX
                MOV DX,0760H
                MOV AH,31H   ;exit and remain resident
                INT 21H

 ;set up  data
      ; ds  =  es     BUFF_G--) pointer data  2 byte
      ; si  =  x1              X1            2 byte
      ; di  =  y1              Y1            2 byte
      ; cx  =  x2     start --) X1           2 byte
      ; ax  =  y2              Y1            2 byte
      ; dx  =  speed  to  mov

START_MOV:      JMP SHORT SET_PARA

 FLAG_C         DW     0
 X1_            DW     0
 Y1_            DW     0
 X2_            DW     0
 Y2_            DW     0
 SPEED_MOV      DW     0
 WIDE           DB     0
 HIGH_          DB     0

 SET_PARA:      MOV DX,03BFH
                MOV AL,3
                OUT DX,AL   ;bring page1 into memory
                CLD
                MOV CX,CS
                MOV DI,OFFSET SAVE_SP
                MOV DX,ES
                MOV ES,CX   ;get new extra segment
                MOV AX,SP
                STOSW       ;save old stack pointer
                MOV AX,SS
                STOSW       ;save old stack segment
                MOV SP,7540H   ;stack pointer = 7540h
                MOV SS,CX      ;get new stack segment
                PUSH DI        ;save old extra segment
                PUSH BP        ;save old base pointer
                PUSH DS        ;save old data segment
                MOV DS,CX
                PUSH ES
                PUSH DS
                MOV AX,0B800H
                MOV ES,AX
                MOV AH,0B0H
                MOV DS,AX
                MOV CX,4000H
                XOR SI,SI
                MOV DI,SI
                REP MOVSW  ;copy image from page0 to page1
                POP DS
                POP ES

                MOV DI,OFFSET BUFF_G
                MOV AX,DI
                ADD AX,6
                STOSW  ;save address buff_g to 2 FIRST BYTE of buff_g
                MOV SI,OFFSET X1_
                LODSW                   ;read  x1
                MOV DX,AX               ;save  x1  to  DX
                LODSW                   ;read  y1
                MOV DI,AX
                LODSW                   ;read  x2
                MOV CX,AX
                LODSW                   ;read  y2
                MOV BX,AX            ;save  y2
                LODSW                   ;read  speed_mov
                PUSH AX      ;************ SAVE  speed_mov ***********
                MOV SI,DX               ;give  X1  to  SI
                MOV AX,BX               ;return  y2
                PUSH DI      ;************ SAVE  Y1 position *********
                MOV DX,4646H
                MOV BX,4747H
                SUB AX,DI
                JG SHORT LIN0
                NEG AX
                MOV BX,4F4FH
LIN0:           SUB CX,SI
                JG SHORT LIN1
                NEG CX
                MOV DX,4E4EH
LIN1:           CMP AX,CX
                JL  SHORT CCC
                XCHG DX,BX
                XCHG AX,CX
CCC:            MOV DI,OFFSET LIN6
                MOV [DI],DX
                MOV [DI + 4],AX
                MOV BP,CX
                MOV [DI + 12],BX
                POP DI      ;return   Y1
                XOR DX,DX
                SHL SI,1
                SHL DI,1
                ADD SI,OFFSET TABLEX
                ADD DI,OFFSET TABLEY
                MOV BX,OFFSET BUFF_G  + 2
                MOV AX,[SI]
                MOV [BX],AX       ;SAVE X1
                MOV [BX + 2],DI   ;SAVE Y1
                POP AX            ;return   SPEED_MOV

LINX:           ROR AX,1
                JNB SHORT LIN6
                MOV BX,[SI]
                PUSH AX
                PUSH SI
                PUSH DI
                MOV SI,OFFSET BUFF_G
                LODSW       ;read  pointer to  save  value      →
```

```
                MOV DI,AX   ;store  pointer  to  DI
                MOV AX,BX   ; x
                STOSW       ;SAVE X
                POP AX      ;return  Y
                STOSW       ;SAVE Y
                MOV [SI - 2],DI  ;SAVE POINTER
                MOV DI,AX
                POP SI      ;***** DATA  FORMAT IN  BUFF_G *****
                POP AX      ; 1)  page_on_screen   0  or 1    |
                            ; 2)  page_segment                |
LIN6:           INC SI      ; 3)  position   of   X           |
                INC SI      ; 4)  position   of   y           |
LIN7:           ADD DX,1234 ;*************************************
                CMP DX,BP
                JL  SHORT LINB
                SUB DX,BP
LINA:           INC DI
                INC DI
LINB:           LOOP LINX
                MOV SI,OFFSET BUFF_G + 2
                MOV DI,OFFSET BUFF_G + 6
                LODSW
                STOSW
                LODSW
                STOSW
                MOV SI,OFFSET BUFF_G
                LODSW
                SUB AX,4
                MOV DI,AX
                MOV SI,OFFSET X2_
                LODSW       ;load X2
                SHL AX,1
                MOV BX,AX
                MOV AX,[BX + OFFSET TABLEX]
                STOSW
                LODSW       ;load Y2
                SHL AX,1
                ADD AX,OFFSET TABLEY
                STOSW
;*************** SET UP DATA  TO  STACK ****************

                MOV SI,OFFSET BUFF_G
                LODSW
                MOV DI,SI
                STD         ;DI is  check  end of process
                MOV SI,AX   ;di is  pointer to first data
                SUB SI,2    ;SI is  pointer to  last data
                MOV AX,0FFFFH
                PUSH AX     ;mark  end  of  data

                ;***** data for write  NEW  IMAGE ******
                MOV BX,008AH
                MOV DX,0B800H
PUSH_CONT:      XOR BX,0080H
                MOV AX,BX               ;save page
                PUSH AX
                LODSW                   ;read y
                PUSH AX
                LODSW                   ;read x
                PUSH AX

                ;***** data for delete  OLD  IMAGE *****
                MOV AX,[SI - 4]         ;read old y
                PUSH AX
                MOV AX,[SI - 6]         ;read old x
                PUSH AX
                XOR DX,00800H
                MOV AX,DX
                PUSH AX       ;save segment   of  page0
                MOV AX,SI
                SUB AX,6
                CMP AX,DI
                JNZ SHORT PUSH_CONT
                XOR BX,0080H
                MOV SI,OFFSET START_PAGE + 1
                MOV [SI],BL
;*************** GIVE   HIGH & WIDE *****************
                CLD
                MOV SI,OFFSET WIDE
                LODSW  ;read high & wide of image
                MOV DI,OFFSET HIGH_WIDEG + 1
                STOSW  ;give high & wide to GET IMAGE
                MOV DI,OFFSET HIGH_WIDEP + 1
                STOSW  ;give high & wide to PUT OLD IMAGE
                MOV DI,OFFSET HIGH_WIDEP1 + 1
                STOSW  ;give high & wide to PUT NEW IMAGE
;get image  from  screen  page0  and  store  to  buff_g
;input  data   si  =  x1
;              di  =  y1
                PUSH DS
                MOV AX,DS
                MOV ES,AX
                MOV BX,OFFSET X1_
                MOV SI,[BX]
                MOV DI,[BX + 2]
                SHL SI,1
                MOV CX,[SI + OFFSET TABLEX ]
                MOV BL,CH
HIGH_WIDEG:     MOV DX,1111H
                MOV CH,DL
                XOR BH,BH
                MOV BP,BX
                MOV BX,DI
                SHL BX,1
                ADD BX,OFFSET TABLEY
                MOV AX,0B000H
                MOV DS,AX
                MOV DI,OFFSET BUFF_G
LOOP_GET1:      MOV SI,BP       ;START BYTE  OF  X
                ADD SI,ES:[BX]  ;OFFSET      OF  Y
LOOP_GET:       MOV AH,[SI]
                INC SI
                MOV AL,[SI]
                SHR AX,CL
                STOSB
                DEC DL
                JNZ SHORT LOOP_GET
                MOV DL,CH
                INC BX
                INC BX
                DEC DH
                JNZ SHORT LOOP_GET1
                POP DS
;put  old image
 start_page:    mov al,11H
                mov dx,03b8h
                out dx,al
                pop ax      ;segment  of  display  buffer
read_data_p:    mov es,ax
                pop cx      ;bit shift & offset  of x
                pop di      ;pointer of  address of y
high_videp:     mov dx,1111h  ;get high and wide of image
                XOR BH,BH
                MOV BL,CH
                MOV BP,BX
                MOV CH,DL
                MOV SI,OFFSET BUFF_G
LOOP_PUT0:      ADD BX,[DI]
LOOP_PUT10:     XOR AH,AH
                LODSB
```

รูปที่ 4 (ต่อ)

```
                SHL AX,CL
                XCHG AH,AL
                XOR ES:[BX],AX
                INC BX
                DEC DL
              JNZ SHORT LOOP_PUT10
                MOV DL,CH
                INC DI
                INC DI
                MOV BX,BP
                DEC DH
                JNZ SHORT LOOP_PUT0
;put  new image
                pop cx    ;bit shift & offset  of x
                pop di    ;pointer of  address of y
high_widepl:    mov dx,1111h  ;get high and wide of image
                XOR BH,BH
                MOV BL,CH
                MOV BP,BX
                MOV CH,DL
                MOV SI,OFFSET BUFF_G
LOOP_PUT1:      ADD BX,[DI]
LOOP_PUT11:     XOR AH,AH
                LODSB
                SHL AX,CL
                XCHG AH,AL
                XOR ES:[BX],AX
                INC BX
                DEC DL
              JNZ SHORT LOOP_PUT11
                MOV DL,CH
                INC DI
                INC DI
                MOV BX,BP
                DEC DH
                JNZ SHORT LOOP_PUT1
                mov dx,03b8h
                pop ax
                out dx,al      ;on screen page0  or  page1
                pop ax         ;read  segment
                cmp ax,0ffffh   ;test for exit loop put image
                jnz  short read_data_p
                MOV SI,OFFSET SAVE_SP
                LODSW
                MOV DX,AX
                LODSW
                POP DS
                POP BP
                POP ES
                MOV SS,AX
                MOV SP,DX
                IRET

REPEAT_RUN      DB    "
TITLE_          DB    "
                DB    "
                DB    "
                DB    "
                DB    "
                DB    "
                DB    "
                DB    "
                DB    "
SAVE_SP         DW    1111h
SAVE_SS         DW    2222h
TABLEX          DW    722D   DUP(?)
TABLEY          DW    350D   DUP(?)
BUFF_G          DB   2000H   DUP(?)

HERC            ENDS
                END START
```

```
[redacted] Fast move  already active [redacted]",13,10,7,"$"
                                                  ",13,10
                                                  ",13,10
    Welcome   to   FAST MOVE  image               ",13,10
    By....  Siwa  wongturun                       ",13,10
    Chon-Chai  school                             ",13,10
      Tel  282078 (038) Chonburi                  ",13,10
                                                  ",13,10
                                                  ",13,10,"$"
```

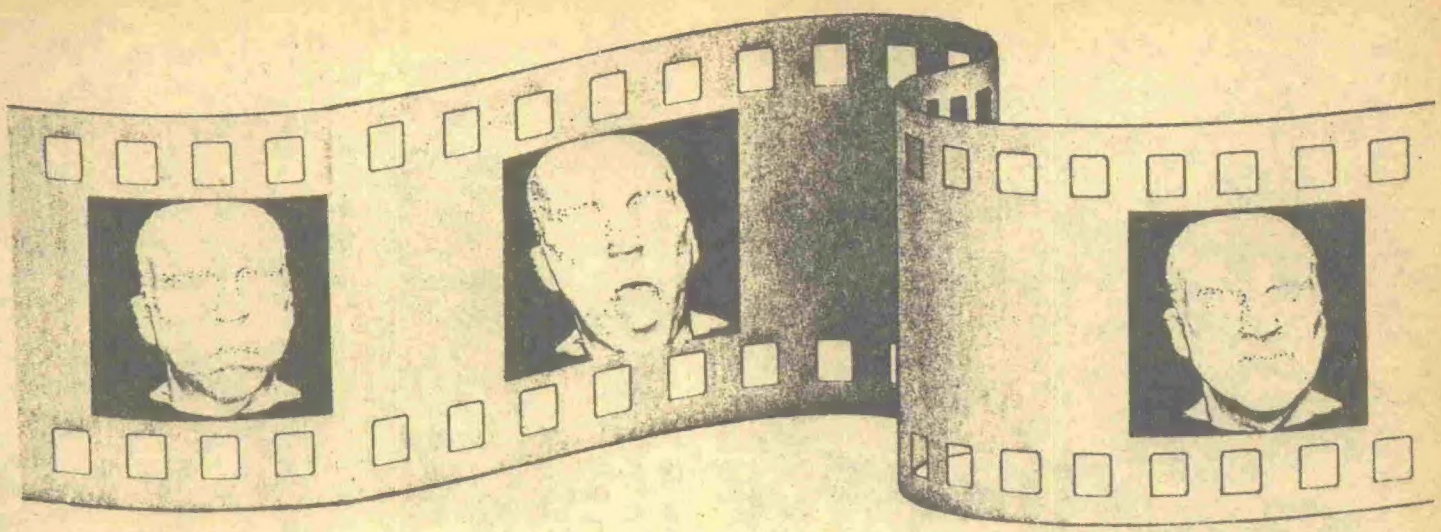
วิธีการสร้างภาพเคลื่อนไหวใน FM.COM เริ่มจากเก็บภาพบนจอภาพในตำแหน่งที่กำหนดไว้แล้วคำนวณหาตำแหน่งระหว่างจุด X1,Y1-X2,Y2 ในการคำนวณหาตำแหน่งจุดนั้นมีค่า SPEED เป็นตัวกำหนดว่าจะแสดงภาพในตำแหน่งระหว่างจุด X1,Y1-X2,Y2 ในที่ใดบ้าง คือดูจากค่า BIT ของ SPEED ที่เป็น 1 ROTATE ผ่าน CARRY FLAG ก็จะเก็บตำแหน่งจุดนั้นไว้พร้อมข้อมูลที่จำเป็นคือ PAGE ที่ ON SCREEN ค่า SEGMENT ที่จะลบภาพเก่า แล้วเขียนภาพในตำแหน่งใหม่ ค่าทั้งหมดนี้จะเก็บลง STACK ด้วยคำสั่ง PUSH แล้วเรียกใช้อีกครั้งเพื่อแสดงภาพโดยคำสั่ง POP

ในการแสดงภาพ ภาพที่มองเห็นบนจอเกิดจากการลบภาพในตำแหน่งเก่าแล้วเขียนภาพในตำแหน่งใหม่ลงไป ขณะที่ลบหรือเขียนภาพนั้น ผู้อยู่หน้าจอภาพไม่สามารถสังเกตเห็นได้ เนื่องจากโปรแกรมจะ SWITCH ไปแสดงภาพในอีก PAGE หนึ่งขณะมีการลบและเขียนภาพใหม่ในอีก PAGE หนึ่งภาพที่ได้จะดูไม่กะพริบ ดั่งการใช้วิธีการ GET และ PUT ภาพแบบที่ใช้กันทั่ว ๆ ไป สำหรับรายละเอียดคงจะต้องดูจากตัวโปรแกรมที่ผู้เขียนได้แสดงไว้แล้ว

FM.COM ทำอะไรบ้างเมื่อเข้าสู่หน่วยความจำ?

แรกสุดก็คือการตรวจสอบเพื่อป้องกันไม่ให้เรียก FM.COM ซ้ำอีกเป็นครั้งที่สอง โดยตรวจสอบจากตำแหน่ง FLAG_C ดั้งเดิมนั้น FLAG_C มีข้อมูล = 0 แต่หลังจากติดตั้งแล้ว ข้อมูลจะเปลี่ยนเป็น 1234H ถ้าพบว่าเป็นการติดตั้งครั้งแรก ในทุกครั้งที่โหลด FM.COM ส่วนที่ตรวจสอบการเรียกซ้ำจะเปรียบเทียบข้อมูลใน FLAG_C กับค่า1234H ถ้าพบว่าเท่ากัน นั่นคือเกิดการเรียกซ้ำขึ้นแล้ว โปรแกรมจะแสดงข้อความพร้อมเสียงเตือนและกลับสู่ดอส สาเหตุที่ป้องกันไม่

■



ให้มีการเรียกซ้ำก็คือ FM.COM เปลี่ยนตำแหน่ง INT 6 มาชี้ที่ตนเองทุกครั้ง แล้วกันเนื้อที่ไว้ 7600H ไบต์ ดังนั้นเนื้อที่นี้จะเพิ่มขึ้นเรื่อย ๆ ถ้าปล่อยให้มีการเรียกซ้ำได้อีกเป็นการสิ้นเปลืองโดยเปล่าประโยชน์

หลังจากผ่านขั้นตอนนี้ไปแล้ว สิ่งที่ทำต่อไปก็คือ สร้างตารางใช้คำนวณค่าตำแหน่งของ X, Y และ BIT SHIFT เพื่อใช้สำหรับการเก็บภาพและแสดงภาพ โดยความจริงแล้ว การคำนวณตำแหน่ง X, Y นั้นมีสูตรใช้อยู่ แต่วิธีการสร้างตารางนี้จะทำให้การเก็บภาพหรือเขียนภาพทำได้รวดเร็วกว่าการคำนวณโดยใช้สูตรทุกครั้งเมื่อจะเก็บหรือเขียนภาพ ตารางที่สร้างขึ้นแบ่งเป็นสองตารางคือ TABLEX เก็บค่าไบต์ที่จุด X อยู่และค่าจำนวน BIT ที่ต้อง SHIFT ตาราง TABLEY เก็บค่าตำแหน่งในหน่วยความจำของ Y ทั้งสองตารางนี้ถูกสร้างขึ้นและคงอยู่ตลอดเวลา เมื่อเสร็จเรียบร้อยก็ใช้ฟังก์ชันของดอสเบอร์ 25H ในการย้ายค่า INT 6 เดิมมาชี้ที่ START__MOV และจบการทำงานโดยกันเนื้อที่ไว้ 760H พารากราฟเป็นอันจบการทำงานเมื่อเริ่มติดตั้ง

สำหรับในส่วนของการทำงาน เมื่อมีการเรียกใช้ INT 6 จะมีขั้นตอนดังต่อไปนี้

1) ต่อ PAGE 1 (B8000-BFFFF) เข้ากับ PAGE 0 (B0000-BFFFF) เก็บค่า SEGMENT REGISTER รวมทั้ง POINTER BP, SP แล้วหันมาใช้ SEGMENT REGISTER ของ CS คือให้ค่า DS, ES, SS เท่ากับ CS ส่วน SP = 7540H

2) COPY ข้อมูลทั้งหมดใน Page 0 ไปที่ Page 1 (32 kB) คำนวณตำแหน่งที่จะแสดงภาพในระหว่างจุด X1,Y1-X2,Y2 เก็บไว้ที่ BUFF__G

3) นำข้อมูลใน BUFF__G เก็บลง STACK พร้อมกับค่า SEGMENT REGISTER, Page

4) เก็บภาพบนจอภาพในตำแหน่งที่กำหนดโดยโปรแกรมเรียกใช้ไว้ใน BUFF__G ซึ่งกันเนื้อที่ไว้ทั้งหมด 2000H หรือ 4096 ไบต์

5) แสดงภาพบนจอโดยอาศัยข้อมูลจาก STACK ทั้งหมด จนกระทั่งพบข้อมูลเป็น 0FFFFH จึงออกจาก LOOP แสดงภาพ

6) คืนค่า SEGMENT REGISTER ของเดิมทั้งหมด แล้วกลับคืนสู่โปรแกรมผู้เรียกใช้ด้วยคำสั่ง IRET

ทั้งหมดนี้คือการทำงานอย่างคร่าว ๆ ซึ่งอันที่จริงยังมีรายละเอียดอื่น ๆ อีกมากมาย ที่ไม่สามารถจะบรรยายได้ สำหรับผู้ที่ใช้ภาษาแอสเซมบลีอยู่แล้ว คิดว่าคงไม่ยากที่จะทำความเข้าใจในโปรแกรมนี้ โดยเฉพาะในส่วนที่ทำหน้าที่คำนวณตำแหน่งระหว่างจุด X1,Y1-X2,Y2 ส่วนนั้นคือส่วนที่ดัดแปลงมาจากโปรแกรมเขียนเส้นตรง และส่วนที่ทำหน้าที่เก็บข้อมูลลง STACK คือ หัวใจของการแสดงภาพเคลื่อนไหว จากแนวทางที่ผู้เขียนพัฒนาโปรแกรมนี้ ผู้อ่านสามารถนำไปพัฒนาต่อไปได้อีกมากมาย เช่น เพิ่มเติมจากการเคลื่อนที่เฉพาะแนวเส้นตรงเป็นเคลื่อนที่ในแนวเส้นโค้งด้วย หรือให้มีการปรับความเร็วโดยกำหนดอัตราเร่ง รวมทั้งภาพเคลื่อนไหวที่มีหลาย ๆ ฉาก ฯลฯ ผู้เขียนหวังว่าจากโปรแกรมนี้คงมีส่วนช่วยให้เกิดความคิดสร้างสรรค์ใหม่ ๆ ขึ้นต่อไป สำหรับผู้อ่านท่านใดซึ่งมีข้อสงสัยเกี่ยวกับโปรแกรม หรือต้องการติดต่อเพื่อแลกเปลี่ยนความคิดเห็น สามารถติดต่อผู้เขียนได้ที่ โรงเรียนชลราษฎรอำรุง อ.เมือง จ.ชลบุรี (20000) หรือ โทร. 282078 (038) ในเวลาราชการ